Homework 7: Q2

Name: Waiwai Kim, waiwaiki

1 Part (a), sub-part 1: Algorithm Idea

Please refer to the week 10 recitation note for a detailed answer.

2 Part (a), sub-part 2: Algorithm Idea

The algorithm idea of the protocol whose worst-case communication cost is $O(nM)$ is pretty similar to part (a). Note that in a star graph, node t is in the middle and each non-t node has an edge to t. Every $u \in V\backslash\{t\}$ sends all of its A_u to t via the edge (u, t). This is possible because every node surrounding node t has an edge to the center node t. Said differently, $MST(G) = n-1$ where G is a star graph. Please refer to the note provided in the homework page. Then t computes the final output. There are total $n-1$ number of surrounding nodes that commit communication to node t. Each node u sends A_u to t only once. Each communication entails sending M bits from $u \neq t$ to t. This implies that each communication incurs cost $1 \cdot M = M$. Since there are $n-1$ nodes committing communication, the overall bounding cost in a star graph is $(n-1) \cdot M$, which is $O(nM)$.

3 Part (b): Algorithm Idea

The algorithm idea is to build a minimum spanning tree based on an input graph G using one of the three algorithms we learned in class. We will use Prim's algorithm to construct a minimum spanning tree. Once we have a minimum spanning tree, we need to figure out the order of traversal or the order of communication. We need to figure this out before sending bits to t node because not every node has a direct edge to t. We will use the BFS algorithm in 3.3 of the textbook. We will run the BFS on node t. Once the BFS algorithm has established the levels of the BFS tree, we can communicate from the leaf nodes to the parents or nodes from level n to level $n-1$ where t sits on level 1. Said differently, we will communicate from the bottom to the top. Here, figuring out the common parents of nodes may take non-optimal computational time; however, this is not a concern because we are only concerned with the algorithm's communication costs. Once a parent node gathers arrays or M bits information from its each children, the parent will run an AND operation on all the arrays that it has received with an array of itself. We will continue this operation for all nodes at a certain level of the BFS tree, after which the algorithm will go to the next level. The algorithm continues until we finish at node t.

4 Part (b): Algorithm Details

Here the input G is an adjacency list expressed as a dictionary of dictionaries. $G[v]$ is a dictionary of edges where node v has edges to. $G[v][u]$ is a distance from node v to node u.

First, we initialize key with infinity. key[node] is used to pick the minimum value in the cut where node is in the the growing MST. MST[u] tells us whether u has been added to the growing MST or not. Parent stores the structured MST.

Secondly, we have a utility function called findMInd. For a given G, key and mst, findMind returns the node that the current MST has the shortest edge to. This is the node to be added based on Prim's algorithm. Once the node is added to the growing MST set, the distances from the node to the nodes outside of the set have to be updated. This is why we have key. Said differently, we have to update the distances of the edges across MST and non-MST or simply known as "Cut". line 12 to 17 in Algorithm 2 perform this update. The algorithm looks the adjacent nodes of $G[min]$, checks if the adjacent node is alrady in the MST or not and if its distance is smaller than the running minimum.

Once the MST is established, the algorithm has to establish the order of communication. Here the algorithm will BFS on s and build a BFS tree with levels denoted as $L[i]$. The line from 27 to 39 is taken from the algorithm presented in 3.3 of the textbook.

From line 42 to 49, the algorithm travel from the bottom layer of the BFS tree to the root or s. The children nodes sends their arrays or bits information to the parent, and the parent will run AND operation. The algorithm will repeat this process until we compute the final bits information at root. Once the root nodes finishes AND operations based on the bits information from all its children nodes, the algorithm can return the output. At this point, the algorithm terminates.

Algorithm 1 findMin

```
1:  procedure FINDMIN(Inputs: G, key, mst)
2:      min = float('infinity')
3:      minNode = None
4:      for v, edgelist in G.items() do
5:          if key[v] < min and mst[v] == 0 then
6:              min = key[v]
7:              minNode = v
8:          end if
9:      end forreturn minNode
10: end procedure
```

5 Part (b): Proof of Correctness Idea

The correctness of the algorithm will be proven over three different parts. First,m we will prove that the algorithm to build a minimum spanning tree is correct. We will do so by proving that Prim's algorithm is correct based on the cut-property. Secondly, we will prove that running BFS on a any tree including a minimum spanning tree allows us to visit every single node. Thirdly, we will prove that AND operation is communicative so that the algorithm doesn't require the starting node or node t to collect bits information form the rest of the nodes in the minimum spanning tree.

6 Part (b): Proof Details

Proving the correctness of Prim's algorithm is shown in 4.5 of the textbook. The proof is based on the cut property. Specifically 4.19 of the textbook claims that Prim's algorithm does indeed produce a minimum spanning tree of G. Secondly, the correctness of BFS is shown in 3.3 of the textbook. The communicative algorithm of AND property is an established property.

Algorithm 2 Algorithm1

```
procedure ALGORIGHM-1(Inputs: s starting node, G as adjacency list, A as list of lists )

    key ={v:float('infinity') for v, _ in G.items() }
    key[s] = 0
    mst = {v:0 for v, _ in G.items() }
    parent ={v:None for v, _ in G.items()}
    parent[s] = -1

    for  v,e in G.items(): do
        min = findMin(G, key,mst)
        mst[min] = 1
        for  vertex, distance in G[min].items(): do
            if mst[vertex] ==0 and distance < key[vertex] then
                key[vertex] = distance
                parent[vertex] = min
            end if
        end for
    end for

    #run BFS on s
    discovered = {v:0 for v, _ in G.items()}
    discovered[s]=1
    i = 0
    initialize L[i] consisting of s
    set the current BFS tree T = non-empty

    while L[i]:  w do
        L[i+1] = []
        for node in L[i]: do
            for  vertex, distance in G[node].items(): do
                if discovered[vertex] == 0: then
                    discovered[vertex] =1
                    add edge(node, vertex) to T
                    v to L[i+1]
                end if
            end for
        end for
        i += 1
    end while

    # communicate from the bottom of T to the top t
    while  i != 0 :  do
        for for u in L[i]: do
            find u's parent $p$
            send $A_u$ to via $(u, p)$
            $A_p = A_u and A_p$
        end for
        i-= 1
    end while
    return $A_p$
end procedure
```

7 Part (b): Bounding Communication Cost

The bounding cost of the algorithm presented above is $O(MST(G) \cdot M)$. Note that we are not concerned with the algorithm's computational cost. First, we know that Prim's algorithm produces a minimum spanning tree based on 4.19. Thus we know that the sum of the distances or the weights of the edges are in MST equals to MST(G). Additionally, we know that every edge needs to transmit M bits exactly once. Thus, we know that $n-1$ edges transmit M bits. Thus we can say that $\Sigma_{i=1}^{n-1} c_i \cdot M$. We can say that $\Sigma_{i=1}^{n-1} = MST(G)$ because we know T is a minimum spanning tree produced by Prim's algorithm. To conclude, the bounding communication cost equals to $O(\Sigma_{i=1}^{n-1} c_i \cdot M) = O(MST(G) \cdot M)$.